API-Shiny Pipeline

Leah Jackman

4/13/2020

Learning Objectives

At the end of this lesson, you should be able to…

- Gain insight into the role of a data scientist in industry
- Interact with simple APIs.
  - Find and read API documentation.
  - Submit a GET request for data.
  - Parse data from the API.
- Create simple Shiny applications.
  - Construct a UI function.
    - Understand the basics of a DOM (Document Object Model).
    - Understand the basic R functions that construct the DOM.
  - Construct a Server function.
    - Understand observe functions.
    - Understand render functions.
    - Understand reactive variables.

The Real World

Part of what a Data Scientist does is…

Find data, assess data, analyze data, visualize data

The Real World: Find Data

Boss: Find the latitude and longitude of all the Walgreens and Boots locations world-wide. When can you get this done?

Me: (Never worked with APIs, maps, geospatial analysis) …

Boss: When can you get this done?

Me: A week?

The Real World: Assess Data

Me: There's an indicator column in this database, but all the values are 0. (Knows there must be at least one 1.) Any idea why?

Database Owner/IT Chatbot: Negative. See. Owner. Of. Table.

Owner of Table/Overworked Project Manager: (shrug)

Me: (Guess I can't trust the data in that column. I wonder about the rest of the table…)

The Real World: Analyze Data

Old School Industry Statistician: Here is a file with thousands of comments from doctors. Use NLP to find all the negative ones.

Me: (Confident, having just read Tidy Text Mining) I can do a sentiment analysis on the comments, shouldn't take me too long!

Me: (Proud, returns with data) Here!

OSIS: Now use NLP to tell me why the doctors are upset.

Me: (Unsure) Well I can do some clustering to see which comments are related and analyze to see if the clusters are meaningful…

OSIS: No, I want quotes from the actual comments that state the exact reason the doctors are upset.

Me: (Looking at data - no labels, just free text)… I don't think NLP can do that …

OSIS: (Frown)

The Real World: Visualize Data

Me: (Demos fancy new data visualization)

Boss: This isn't sexy enough. I saw an application that has a spinny-globe loading screen, can you add that?

Me: Well… Maybe… I can try …But what does that have to do with this visualization?

Boss: SPINNY GLOBE! SEXY! BRING ME!

APIs

- API stands for Application Programming Interface.
- Serves as a mediator/gatekeeper.
- Takes as input a request in a very strict format.
- Request is parsed, and initiates some set of processes.
- Processes return data objects in a structured format - JSON, XML, etc.

Requests

httr package

jsonlite package

GET

Me: (Goes to API's front door. Piles of data with my name on it sitting there.) Thanks API!

API: You're welcome, use the data wisely or you will be sued. Have a lovely day!

GET Example

```r
#This request returns the state data aggregated over all time.

#Libraries
library(httr)
library(jsonlite)

#https://documenter.getpostman.com/view/8854915/SzS8rjHv?version=latest State Level Data
#set a GET request
covid_states_req <- GET("https://covidtracking.com/api/states", query = list("state" = "NY"))

status_code <- covid_states_req$status_code

#"good" request when status = 200.
if (status_code == 200) {
  #get content of request.
  #many ways to do this, but I like the following because the output is a data frame
  data_df <- jsonlite::fromJSON(content(covid_states_req, as = "text"))

  #"good" means correct format of request, but might still be issue with query itself
  #if issue with query, return error message
  if ("error" %in% names(data_df)) {
    error_msg <- data_df$message
    data_df <- NULL

  }else {
    data_df <- data_df
  }
}

head(data_df)
```

You Try! Write a request that returns all data, not aggregated data, for a given state. Hint: Read the documentation

POST

Me: (Goes to API's front door, door closed) Knock, knock. Here's my req-

API: (Interrupting) Intruder! Intruder!

Me: Oh, right right. (Puts eyeball up to optical scanner, completes secret knock knock pattern)

API: Phew! OK, now what do you want?

Me: (Sends list of exacting instructions.)

API: (Scrutinizing) Looks to be in order. Here's your data. Use it wisely or you will be sued. Have a lovely day!

Including an example here, but we're not going to go over it.

POST Example

```r
#' Search Citeline for Protocol Information
#'
#' Searches Citeline Trial endpoint based on user input
#'
#' @param return_fields_lst a list of API fields to return
#'
#' @param search_fields_lst a list of API fields to filter on
#'
#' @param operators_lst a list of API operators to apply to search_field_lst and search_values_lst
#'
#' @param search_values_lst a list values used to filter search_field_lst
#'
#' @param user_name_txt the user name for the search. Should be standard CI credentials.
#'
#' @param password_txt the password for the search. Should be standard CI credentials.
#'
#' @return The first 100 matches in a dataframe. One column per return_field_lst value.
#'
#' @export

search_similar_trials_fn_gbl <- function(return_fields_lst,
                                         search_fields_lst,
                                         operators_lst,
                                         search_values_lst,
                                         user_name_txt,
                                         password_txt,
                                         session){
  library(httr)

  url <- "https://identity.pharmaintelligence.informa.com/connect/token"
  auth_token <-
    "Y3VzdG9tZXJfYXBpX2NsaWVudF9uYXRpdmU6NTIwNzhjMGItMTI5Mi00MGVhLWFkYjAtOWE4MWY4OGNjZDMy"
  authHeader <- add_headers(Authorization = paste("Basic", auth_token))

  post_req <- POST(url,
                   authHeader,
                   body = list(
                     grant_type = "password",
                     scope = "customer-api",
                     username = user_name_txt,
                     password = password_txt))
  post_output <- content(post_req, "parsed")
  token <- post_output$access_token

  api_url <- "https://api.pharmaintelligence.informa.com"
  endpoint <- paste0("/v1/search/trial")
  usage_url <- paste0(api_url, endpoint)

  authHeader <- add_headers(Authorization = paste("Bearer", token),
                          Accept = "application/json")
```

```r
#Note: Do not follow this example! This is before I learned about jsonlite::toJSON.
#All you need to do is create a list of lists and use toJSON
#ex) json_req <- list(and = list(field = "investigator_name", value = "Dr. Person")) %>% toJSON()
start_query <- '\"and\":['
primary_query <- paste0(paste0('{\"', operators_lst, '\": {\"value\": \"', search_values_lst,
                              '\",\"name\": \"', search_fields_lst, '\"}}'), collapse = ",")
end_query <- ']'

query_list <- paste0(start_query, primary_query, end_query)

query <- paste0("{\"fields\": [\"", paste0(return_fields_lst, collapse = "\",\""), "\"],", query_list

post_req <- POST(usage_url, authHeader, body = query, content_type_json())
get_output <- httr::content(post_req, "parsed")

# Error Handling based on get_output's status code
if (get_output$meta$statusCode == 400) {
  error_title <- "Bad Request"
  error_message <- "The request contains invalid parameters, or invalid or incomplete data"
} else if (get_output$meta$statusCode == 401) {
  error_title <- "Unauthorized"
  error_message <- "You failed to provide an authentication header, or your credentials are invalid o
} else if (get_output$meta$statusCode == 403) {
  error_title <- "Forbidden"
  error_message <- "You tried to perform an action that is not authorised. E.g. you failed to request
} else if (get_output$meta$statusCode == 404) {
  error_title <- "Not Found"
  error_message <- "The resource or endpoint does not exist. This could be temporary or permanent"
} else if (get_output$meta$statusCode == 406) {
  error_title <- "Not Acceptable"
  error_message <- "The API does not support the requested format in the Accept header. Try again usi
} else if (get_output$meta$statusCode == 500) {
  error_title <- "Internal Server Error"
  error_message <- "You should never receive this error because our clever developers catch them all.
} else if (get_output$meta$statusCode == 503) {
  error_title <- "Service Unavailable"
  error_message <- "The API has been switched to maintenance mode for a major or complex release"
}

# Display error message
if (get_output$meta$statusCode != 200)
  sendSweetAlert(session = session, title = error_title, text = error_message, type = "error")

# Dealing with pagination
if (length(get_output$items) > 0) {
  # Names of output fields
  output_items <- get_output$items
   next_page <- get_output$pagination$nextPage

   while(!is.null(next_page)){
     get_next_req <- GET(next_page, authHeader)
     get_next_output <- httr::content(get_next_req, "parsed")
     output_items <- append(output_items, get_next_output$items)
```

```
      next_page <- get_next_output$pagination$nextPage
     }
    return(output_items)
  } else {
    return(NA)
  }

}
```

API Tips and Tricks

- Read the documentation!
- Use the verbose() option to troubleshoot.
- Special Characters - either escape them with \ or use I()

Shiny

Shiny is a package in R that constructs websites.

Two ways to construct a website in Shiny

- ui.R, server.R
- ui function, server function together in one .R file.

UI

- Creates the original DOM (Document Object Model)
- Outlines what objects (sidebars, navigation menus, boxes, inputs, etc.) exist, and where they exist in a website.

Demo "inspect" on website to show example.

UI Example

Simple Example from RStudio Uses the two separate .R files approach

```
vars <- setdiff(names(iris), "Species")

pageWithSidebar(
  headerPanel('Iris k-means clustering'),
  sidebarPanel(
    #each input has an associated class
    selectInput('xcol', 'X Variable', vars),
    selectInput('ycol', 'Y Variable', vars, selected = vars[[2]]),
    numericInput('clusters', 'Cluster count', 3, min = 1, max = 9)
  ),
  mainPanel(
    #each output has an associated class
    plotOutput('plot1')
  )
)
```

You Try! What other types of inputs exist beside select and numeric?

Server

- Tells the website how it should behave based on interactivity.
- For the most part, hidden from user, so "Inspect" not as helpful.
- Input variables from the ui can be accessed using "input$*input id*"
- Input variables from the ui cannot be accessed, but can be defined using "output$*output id*"

Server Example

Simple Example from RStudio Uses the two separate .R files approach

```
function(input, output, session) {

  # Combine the selected variables into a new data frame
  # selectedData is available to every function in the server, and will automatically change if xcol or
  # note the ({}) syntax - this is use for all reactivity - renderers, observers, reactive variables, e
  selectedData <- reactive({
    #inputs accessed here
    iris[, c(input$xcol, input$ycol)]
  })

  #instead of reactive, can also use observe(), observeEvent() depending on needs
  clusters <- reactive({
    kmeans(selectedData(), input$clusters)
  })

  #setting the output object
  output$plot1 <- renderPlot({
    palette(c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3",
      "#FF7F00", "#FFFF33", "#A65628", "#F781BF", "#999999"))

    par(mar = c(5.1, 4.1, 0, 1))
    plot(selectedData(),
         col = clusters()$cluster,
         pch = 20, cex = 3)
    points(clusters()$centers, pch = 4, cex = 4, lwd = 4)
  })

}
```

You Try! Many packages other than ggplot2 exist to create visualizations in R. Find one such package How would you amend the above code to use the new package?

COVID-19 Example

GitHub Project

Download and play around. Some things to try:

- Make the graph prettier.
- Add a select input for model types - Exponential, Logistic, etc. - that would be appropriate for this data. When the user selects a model, overlay the model on the graph.
- Check out shinyDashboard and shinyDashboardPlus and try to add a theme to make the website more appealing.
- Add a checkbox with "confirmed" and "deaths". Use the checkbox to allow the user to see either confirmed, or deaths or both on the graph.
- Change the country select input to a selectizeInput where you can accept multiple values. Plot the curves for all countries on a single graph.
- Add in states/provinces for countries other than the US.

Reflection

At the end of this lesson, you should be able to…

- Gain insight into the role of a data scientist in industry
- Interact with simple APIs.
  - Find and read API documentation.
  - Submit a GET request for data.
  - Parse data from the API.
- Create simple Shiny applications.
  - Construct a UI function.
    - Understand the basics of a DOM (Document Object Model).
    - Understand the basic R functions that construct the DOM.
  - Construct a Server function.
    - Understand observe functions.
    - Understand render functions.
    - Understand reactive variables.